15

AIRMICS

Army Institute for Research in Management Information and Computer Science

313 Calculator Bldg.
GA Institute of Technology
Atlanta, GA 30332

Technical Report

RESEARCH IN FUNCTIONALLY DISTRIBUTED COMPUTER SYSTEMS DEVELOPMENT

Kansas State University

Virgil Wallentine
Principal Investigator

VOLUME IX

MEMORY MANAGEMENT IN A DISTRIBUTED DATA BASE MANAGEMENT SYSTEM

U.S. ARMY COMPUTER SYSTEMS COMMAND FT BELVOIR, VA 22060

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

REPORT DOCUMENTATION PAGE		READ INSTRUCTIONS BEFORE COMPLETING FORM
1. REPORT NUMBER	2. GOVT ACCESSION NO. AD-A103 074	3. RECIPIENT'S CATALOG NUMBER
4. TITLE (and Subtitle) MEMORY MANAGEMENT IN A DISTRIBUTED DATA BASE MANAGEMENT SYSTEM.		5. TYPE OF REPORT & PERIOD COVERED Iterim
		6. PERFORMING ORG. REPORT NUMBER CS 76-14
7. AUTHOR(s) Fred Maryanski		8. CONTRACT OR GRANT NUMBER(s) DAAG 29-76-G-0108
9. PERFORMING ORGANIZATION NAME AND ADDRESS Kansas State University Department of Computer Science Manhattan, KS 66506		10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS
11. CONTROLLING OFFICE NAME AND ADDRESS US Army Research Office P O Box 12211 Research Triangle Park, NC 27700		12. REPORT DATE October 1976
		13. NUMBER OF PAGES 48 pages
14. MONITORING AGENCY NAME & ADDRESS (if different from Controlling Office) US Army Computer Systems Command Attn: CSCS-AT Ft. Belvoir, VA 22060		15. SECURITY CLASS. (of this report) Unclassified
		15a. DECLASSIFICATION DOWNGRADING SCHEDULE

16. DISTRIBUTION STATEMENT (of this Report)

Approved for public release; distribution unlimited.

17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)

18. SUPPLEMENTARY NOTES

The findings in this report are not to be construed as an official Department of the Army position, unless so designated by other authorized documents.

19. KEY WORDS (Continue on reverse side if necessary and identify by block number)

DDBMS
Distributed Processing
Back-end Computer

20. ABSTRACT (Continue on reverse side if necessary and identify by block number)

-over-

DD FORM 1 JAN 73 1473 EDITION OF 1 NOV 65 IS OBSOLETE

Unclassified

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

-ABSTRACT-

A memory management scheme which incorporates an additional level of memory into the traditional primary-secondary storage hierarchy is proposed for utilization in distributed data base management systems. In this scheme, the memory of the back-end processor is used as an additional memory buffer. An optimal three-level memory management algorithm is presented along with an analysis of its cost in terms of page replacement. The expected performance improvement over the optimal algorithm for a two-level memory system is determined. The performance benefits of the three-level memory management are applicable to most distributed processing systems.

Memory Management in Distributed Data Base Systems*

Technical Report CS 76-14

by

Fred J. Maryanski
Computer Science Department
Kansas State University
Manhattan, Kansas 66506

October 1976

A

* The work reported herein is supported by the United States Army Research Office, Grant No. DAAG 29-76-G -0108.

ABSTRACT

A memory management scheme which incorporates an additional level of memory into the traditional primary-secondary storage hierarchy is proposed for utilization in distributed data base management systems. In this scheme, the memory of the back-end processor is used as an additional memory buffer. An optimal three-level memory management algorithm is presented along with an analysis of its cost in terms of page replacement. The expected performance improvement over the optimal algorithm for a two-level memory system is determined. The performance benefits of the three-level memory management are applicable to most distributed processing systems.

1. INTRODUCTION

The primary goal of data base management systems is to provide rapid and secure processing of large amounts of data. Through the use of a data base management system (DBMS) data can become easily available to a large class of people ranging from the data base administrator who has specified the logical and physical structure of the data base to the clerk who enters requests on a keyboard. Data base systems have evolved to the point that programs can be written to perform virtually any type of operation on a data base. There are a large number of commercially available general data base systems [1].

A common characteristic of present day state of the industry systems is that the data base is under the control of a single computer. The ability to operate on data controlled by several distinct computer systems is the next logical step in the evolution of data bases. A system in which the data bases controlled by physically separated processors are accessible to all processors is known as a distributed data base management system.

The idea of a DBMS operating in a multi-computer environment has been discussed by several authors [2-7]. Canaday, et al. [5] developed a prototype backend DBMS. A back-end DBMS is a two processor configuration in which one machine (the host) executes application DBMS programs and the second machine (the backend) performs the actual data base operations upon request from the host. In a back-end DBMS, control of the data base resides in the back-end processor.

The feasibility of a back-end DBMS in a data processing environment has been investigated in a study reported in Reference [9]. The results of the study indicate that a back-end DBMS frees host CPU and memory resources, introduces concurrency into the system, and provides an economical means of increasing system capacity.

This paper proposes a three-level memory management scheme for distributed data base systems. This scheme which employs the back-end memory as an additional buffer between the application program and secondary storage is analyzed in terms of cost of page replacement. The projected performance inprovement using the three-level technique is then presented.

2. DISTRIBUTED DBMS FUNCTIONAL CHARACTERISTICS

Essentially a distributed DBMS is a data management facility which resides on a computer network each of whose nodes has three capabilities with respect to data management. Three functions of a processor node in a DDBMS network are listed below.

1. User interface. Serve as job initiation point and input/output facility.
2. Application program execution. The data base application program resides in the memory of and is executed by this processor.
3. Data base access. Each node controls the access to the data base residing on the secondary storage devices connected to the processor.

In a DDBMS any application program may be submitted to one processor, executed on another, and have access to data bases on any other nodes in the network. An important feature of a distributed DBMS is that the processor used to execute the program and the physical location of the data may be totally transparent to the user. The application program may reference any data item to which it has legal access by a logical name, the mapping to the physical location in the network is carried out by the DBMS.

A typical distributed DBMS topology is depicted in Figure 1. The front-end processors are used exclusively for user interface; the host machines are dedicated to application program execution and a back-end computer's sole function is data base access. In the general case, any machine in the distributed DBMS may be assigned any combination of the three data base functions. The only restrictions are that a front-end machine be interfaced to one or more terminals; a host must have sufficient memory to execute the application program and a multi-programmed operating system, (approximate minimum-32Kb); the back-end processor must support multi-programming, have direct and sequential access capabilities and be directly tied to the secondary storage devices containing the data base, and have sufficient primary memory to support the DBMS function (at least 64Kb).

3. MEMORY MANAGEMENT

Except when the processors in the distributed DBMS network are physically proximate with ultra high speed links, intermachine transmission time becomes a limiting performance factor. Therefore the frequency of large scale data transmissions between machines must be minimized. In the situation of very high speed intermachine connection, disk access time becomes an important consideration. Here limiting the frequency of data transfers from disk to memory increases system performance.

Since the distributed DBMS concept supports any type of machine connection, a generalized memory management scheme is necessary to minimize data transfers. Three levels of memory are available to each application program, host, back-end, and secondary. The host and back-end memories each contain maps of the pages currently residing within their memories.

When a page is removed from the host memory, a test must be made to determine if it has been modified since being retrieved from the back-end. If not, it is merely overwritten and no transmission to the back-end takes place. If the page to be replaced has been updated, it is returned to the back-end machine with a flag set to indicate that it has been modified. The back-end computer does not return this page immediately to secondary memory, but rather retains it in its primary memory. Thus the back-end retains pages that have been previously used by the application program. Such pages have a higher probability of being accessed than previously unaccessed pages, due to the principle of locality [10,11]. This scheme is, in effect, a three-level page replacement algorithm. If a page is returned to the back-end memory, the original copy if overwritten. When a back-end page is to be replaced, again its write flag is checked and it is written back to the disk only if it has been modified.

This memory management scheme presumes a single back-end machine per physical device. That is, all access to the device must pass through that back-end machine. This eliminates contention problems for that device. Since the back-end by definitien is an I/O oriented processor, any bottle necks in the system would be the result of poor data set distribution on secondary storage devices. This problem can be detected and alleviated by proper use of data base utilities which provide usage statistics and those which restructure the data base.

The following simple example illustrates the benefits of a three-level memory management system.

Example 1

Assume a distributed DBMS configuration of three hosts, H_1, H_2, H_3 and a back-end, B, which controls access to record, R. H_1 and H_2 may both read and write R while H_3 has only read privileges.

Assume the following sequence of actions occurs:

Read R by H_1

Write R from H_1

Read R by H_2

Write R from H_2

Read R by H_3

Read R by H_1

Terminate

If a standard two-level memory management approach were taken (i.e. no buffering in the back-end), the set of data transfers involving R shown in the first column would result. The second column indicates the transfers in a three-level management scheme under optimal conditions.

Operation	2-Level	3-Level
1. Read R by H_1	disk to B, B to H_1	disk to B, B to H_1
2. Write R from H_1	H_1 to B, B to disk	H_1 to B
3. Read R by H_2	disk to B, B to H_2	B to H_2
4. Write R from H_2	H_2 to B, B to disk	H_2 to B
5. Read R by H_3	disk to B, B to H_3	B to H_3
6. Read R by H_1	disk to B, B to H_1	B to H_1
7. Terminate		B to disk

In this example, both techniques required one transfer between the back-end and a host for each operation. The 2-Level approach also requires one disk transfer per operation while the 3-Level method resulted in a total of two disk transfers. The assumption was made that no back-end page fault forced K to be written onto the data base during the sequence of operations. In general, the performance of the memory management scheme is directly related to the number of page faults in the back-end.

In the optimal case, assuming the two-level scheme required K disk transfers, no back-end page faults will occur and a total of K-2 disk transfers will be saved. (K-1 transfers if no writing takes place). The worst case behavior of the three-level memory scheme is identical to the two-level arrangement. In this situation, a page fault occurs before the next request for a given page is made. This completely eliminates the buffering effect of the back-end memory. Eq. (1) gives the reduction in secondary storage transfers for a page in the three-level management environment as opposed to a two-level scheme.

$$P_t = K-2*pfw - pfr - W_r - 1 \qquad (1)$$

where

K is the number of secondary storage transfers for a given page in the two-level scheme;

pfw is the number of times a back-end page fault replaces p when the write flag of p is set;

pfr is the number of times a back-end page fault replaces p when write flag of p is not set; and

W is 1 if upon closing of the file containing p, the write flag is set. Otherwise W is 0.

The reason for the factor of 2 being associated with pfw is that replacement of p with the write flag set results in two operations writing p and then reading it back in for the next access. If the write flag is not set, p is not written back to secondary storage. The next access of p requires only that p be reread. Therefore, pfr is multiplied by a factor of 1 in eq.(1).

From eq. (1) it can be seen that the two factors dominating performance of the three-level memory management policy are the number of page faults and the frequency of write operations.

4. PAGE REPLACEMENT ALGORITHM

AS indicated previously the page replacement algorithm on the back-end machine is a critical performance factor. Therefore, a theoretically optimal algorithm is presented. The following discussion is similar to that found in Reference [12]. The principle difference is in the cost functions.

As in Reference [12] we will assume a 0th order stationary program. This means that p(x), the probability of page x being referenced is independent of previous references and remains fixed throughout the program.

Definition 1 (Terminology)

Let $N = (i,\ldots n)$ be the pages of a given program and $M = (1, \ldots m)$ be pages of back-end memory. Assume that $1 \leq m < n$. N^k comprises all strings of length k over N, $k > 0$. Reference string $v = r_1 \ldots r_k \in N^k$. $P(x,t)$ is the probability that page x will be referenced at time t. $r_t\ x$ implies that a time t, the program references page x.

$S \subset N$ is a memory state.

$|X|$ denotes the number of elements in set X.

W_y is the write flag of page y.

Definition 2

The allocation map, g_A, of paging algorithm A is $g_a(S,x) = S'$

where S, S' are memory states.

x is a referenced page.

Definition 3

A is a demand paging algorithm if the allocation map of A is defined as follows:

$$g_a(S,x) = \begin{cases} S & x \in S \\ S+x & x \notin S,\ |S|<m \\ S=x-y & x \notin S,\ |S| \leq m,\ y \in S \end{cases} .$$

Definition 4

The cost of algorithm A with memory state S and reference string v is

$$C(A,S,v) = \sum_{t=1}^{T} h(y_t) \text{ where } y_t \text{ is the page replaced.}$$

The expected cost of algorithm A with memory state S over all reference strings of length K is

$$C_k(A,S) = \sum_{v \in N^k} p(v)C(A,S,v) .$$

Definition 5

The cost of replacing a page, y, on the back-end machine is

$$h(y) = \begin{cases} 2 \text{ if } W_y \\ 1 \text{ if } \bar{W}_y \end{cases} .$$

It is this cost function that differs from the work in Reference [12]. Aho, Denning, and Ullman did not consider the cost of removing a page in their analysis. Due to the operation of the three-level memory management scheme, it is necessary to include the cost of page removal. Thus, the replacement cost is two disk operations (one each for replacement and removal) if the write flag is set.

Definition 6

The minimum achievable expected cost for processing k references beyond time t_i is $C_k(S,t)$ which is defined recursively as

$$C_o(S,t) = 0$$

$$C_k(S,t) = \sum_{x \in N} p(x,t+1) * \begin{cases} C_{k-1}(S,t+1), & x \in S \\ h(y_t) + \min\limits_{y \in S} C_{k-1}(S+x-y), & x \notin S \end{cases} .$$

$C_k(S,t)$ is the minimum demand paging cost of processing $r_k \ldots r_{k+t}$.

Definition 7

Let < be a ranking relation on N such that if $x<y$ then

$h(x)p(x) < h(y)p(y)$.

$s = \min S$ implies that for $s \in S$, $s \leq x$, $\forall x \in S$.

Lemma 1 (Aho, Denning, and Ullman [12])

For $t>0$ and $\forall S' \subset N$, if $x<y$ implies that $C_k(S'+x,t) \leq C_k(S'+y,t)$, then $s = \min S$ implies that

$$C_k(S-s,t) = \min_{z \in S} C_k(S-z,t) \ \forall S \subset N.$$

Proof

If $s<z$, let $x=s$, $y=z$, $S'=S-s-z$,

then

$$C_k(S-s,t) \leq C_k(S-z,t) \quad \forall z \neq s \in S.$$

Thus

$$C_k(S-s,t) = \min_{\in S} C_k(S-z,t) \quad \forall S \subset N.$$

Lemma 2

Suppose $<$ is a stationary ranking of N.

Then

for $x<y$, $h(y)p(y) \geq C_k(S+y,t) - C_k(S+x,t) \geq 0$, $x,y \in S$.

Proof

An intuitive proof is provided here. A formal proof can be obtained by using the mechanism given in Reference [12] and substituting h(y)p(y) in place of 1 as the upper bound.

Intuitively it can be seen that the only circumstances under which $C_k(S+y,t) - C_k(S+x,t)$ is nonzero is when either x or y must be replaced. If $x<y$ then x has a lower expected cost than y. Thus the difference of the expected minimum costs is positive.

Lemma 3

The optimal back-end page replacement algorithm A_b has the map

$$g_{A_b}(s,x) = \begin{cases} S & x \in S \\ S+x-s & x \notin S \end{cases}$$

where $s = \min S$.

Algorithm A_b replaces the page with the lowest expected cost.

Proof

By Lemmas 1 and 2, A_b makes the minimal cost decisions as indicated by Def. 7. Therefore A_b is a minimal expected cost algorithm.

Lemma 4

If the 0th - order page reference probabilistics are stationary, the expected cost per reference from state S is

$$C(S) = (B - \sum_{i=m}^{n} p^2(i)/B)\ (1 + \sum_{i=m}^{n} p(i)p(W_i))$$

where $B = \sum_{i=m}^{n} p(i)$.

Proof

Let $a_x(t)=p(x/S_t)$ be the probability of a reference to x at time t causing a page fault.

Let $s_t = \text{Min } S_t$ be the lowest ranked page at time t.

Let S_o be the initial memory state. The expected cost of k references given initial state S_o is

$$C_k(S_o) = \sum_{t=1}^{k} p(y_t \epsilon S_{t-1})\quad h(s_t)$$

$$C_k(S_o) = \sum_{t=1}^{k} \left(\sum_{x \epsilon N} p(x) a_x(t-1) \right) h(s_t) \quad .$$

Under algorithm A_b,

$$a_x(t) = \begin{cases} 0 & 1 \le x < m \\ 1 - p(x)/B & m \le x \le n \end{cases}$$

then

$$C_k(S_o) = \sum_{t=1}^{k} (B - (\sum_{i=m}^{n} p^2(i))/B)\ h(s_t) \quad .$$

In general the cost of replacing page s_t can be expressed as

$$h(s_t) = 1+E[W_{s_t}] \quad .$$

Where

$E[W_{s_t}]$ is the expected value of the write flag of page s_t.

$$E[W_{s_t}] = \sum_{i=m}^{n} p(i)\ p(W_i).$$

For notational convenience let

$$F = B - \left(\sum_{i=m}^{n} p^2(i)\right)/(B)$$

then,

$$C_k(S_o) = \sum_{t=1}^{k} F(1+E[W_{s_t}])$$

$$= \sum_{t=1}^{k} F\left(1+\sum_{i=m}^{n} p(i)\ p(W_i)\right)$$

$$= F \sum_{t=1}^{k} 1+F\sum_{t=1}^{k} \sum_{m=i}^{n} p(i)p(W_i)$$

$$=kF \left(1+\sum_{i=m}^{n} p(i)p(W_i)\right) \quad .$$

The expected cost per reference is then

$$C(S_o) = \lim_{k\to\infty} \frac{C_k(S_o)}{k}$$

$$= F\left(1+\sum_{i=m}^{n} p(i)p(W_i)\right)$$

$$= \left(B - \sum_{i=m}^{n} p^2(i)/B\right) \left(1+\sum_{i=m}^{n} p(i)\ p(W_i)\right) \quad .$$

Lemma 4 indicates that the cost of the algorithm A_b is dependent upon the frequency of update operations on the lowest ranked pages. This result differs from the expected cost for the optimal algorithm for 2-level storage, Ao, [12,13]. It should be noted from the definition of the page ranking that in order for a page with its write flag set to be replaced its reference probability must be half that of the lowest page with a cleared write flag.

Example 2

Let n=5, m=3 , N=(a,b,c,d,e)

Assume the following set of probability measures for the pages

$p(a) = \frac{3}{8} \quad p(b) = \frac{1}{4} \quad p(c) = \frac{3}{16} \quad p(d) = \frac{1}{8} \quad p(e) = \frac{1}{16}$.

Assume the following cost functions

h(a)=2 h(b)=1 h(c)=1 h(d)=2 h(e)=1 .

Thus

$h(a)p(a)=\frac{3}{4} \quad h(b)p(b)=\frac{1}{4} \quad h(c)p(c)=\frac{3}{16} \quad h(d)p(d)=\frac{1}{4} \quad h(e)p(e)=\frac{1}{16}$.

The page are thus ordered by <, as

[a,b,d,c,e] .

Assume the follwing reference string

v = abcdeabc .

At	t=0	,	S=∅	
	t=1	,	S=[a]	
	t=2	,	S=[a,b]	
	t=3	,	S=[a,b,c]	
	t=4	,	S=[a,b,d]	, replace c, h(c) = 1
	t=5	,	S=[a,b,e]	, replace d, h(d) = 2
	t=6	,	S=[a,b,e]	
	t=7	,	S=[a,b,e]	
	t=8	,	S=[a,b,c]	, replace e, h(e) = 1

Total cost of v is h(c)+h(d)+h(e) = 4 .

5. PERFORMANCE IMPROVEMENT

The page replacement algorithm A_b, is applicable to both host and back-end processors in a distributed DBMS. The performance of a three-level memory system is dependent upon *the type of connection* between the host and the back-end. If they are tightly-coupled as we have thus far assumed, the back-end memory is essentially an extension of the host memory. However, in the case of a remote connection, the transmission time between host and back-end machine could easily surpass that of a disk access. In a host back-end environment the expected cost of replacing a page is given by

$$C = C_h(S_h) * T + C_b(S_b) * D \qquad (2)$$

where

$C_h(S_h)$ is the expected cost of replacing a page in the host in state S_h;

$C_b(S_b)$ is the expected cost of replacing a page in the back-end in state S_b;

T and D are weighing factors for transmission and disk access times.

It is difficult to fairly compare performance of a distributed DBMS and a single machine system. If the host/back-end connection is slower than shared primary memory then there exists a tradeoff of increased access and communications overhead. If a shared memory linkage is assumed (T=1 in eq.(2)) then the improvement in terms of expected cost of the 3-level management scheme over the 2-level management scheme can be computed.

Let $M_h = \{1 \ldots m_h\}$ be the pages of host memory.

$M_b = \{1' \ldots m_b'\}$ be the pages of back-end memory

and $M = \{m_i | m_i \in M_h \cup \{M_b - M_h\}\}$

$= \{1 \ldots m\}$, note that $m > m_h$.

Let $B = \sum_{i=m}^{n} p(i)$

and $B_h = \sum_{i=m_h}^{n} p(i)$.

Then the expected cost improvement of the three-level memory scheme over the two-level scheme is

$$C_I = C_h(S) - C_b(S)$$

$$C_I = \left(B_h - \sum_{i=m_h}^{n} p^2(i)/B_h\right)\left(1 + \sum_{i=m_h}^{n} p(i)p(W_i)\right)$$

$$- \left(B - \sum_{i=m}^{n} p^2(i)/B\right)\left(1 + \sum_{i=m}^{n} p(i)p(W_i)\right) .$$

Example 3

Consider the system of the previous example, letting $m_h = 3$, $m = 4$.

The expected cost for a host-only system is

$$C_h(S) = \left(\sum_{i=3}^{5} p(i) - \sum_{i=3}^{5} p^2(i)/\sum_{i=3}^{5} p(i)\right)\left(1 + \sum_{i=3}^{5} p(i)p(W_i)\right)$$

$$= \left(\frac{3}{8} - \frac{7}{128} \Big/ \frac{3}{8}\right)\left(1 + \frac{1}{8}\right)$$

$$= \frac{33}{128} = 0.258 .$$

The expected cost for the distributed system is

$$C_b(S) = \left(\sum_{i=4}^{5} p(i) - \sum_{i=4}^{5} p^2(i) / \sum_{i=4}^{5} p(i) \right) \quad 1 + \sum_{i=4}^{5} p(i)p(W_i)$$

$$= \left(\frac{1}{4} - \frac{5}{128} / \frac{1}{4} \right) \left(1 + 0 \right)$$

$$= \frac{3}{32} = .094 \quad .$$

The expected improvement in performance is

$$C_I = C_h(S) - C_b(S) = \frac{33}{128} - \frac{3}{32}$$

$$= \frac{21}{128} = .164 \quad .$$

This indicates that for every six page references, the three-level scheme is expected to have one less disk reference.

6. FEASIBILITY OF ALGORITHM

The memory management algorithm presented here is theoretically optimal. However, it is dependent upon a knowledge of a page reference probability. In a data base environment, a record of all operations is maintained on a journal file for backing and recovery purposes. Page reference probabilities can be computed from the journal file in a straight forward manner. The value of dynamically performing such computations at run time is questionable. However, in a reasonably stable environment, such as the daily cycle of a data processing installation, it should be feasible to periodically compute fairly accurate page reference probabilities which could be used to drive the memory management algorithm.

In a stable environment a page reference model describable by a higher order Markov process could be synthesized. If such a model were realized, an efficient pre-paging scheme [14] would become feasible. Any attempt at implementation of such a model must be proceded by considerable analytical study and careful simulation modeling.

7. CONCLUSION

The results presented in this report indicate that a three-level memory management scheme will provide performance benefits in a distributed data base management system. While the analysis given here concentrated upon theoretically optimal algorithms for page replacement, the effect of the additional buffering in the back-end memory would yield improvement for any algorithm. The three-level memory management concept is applicable to any multi-computer configuration.

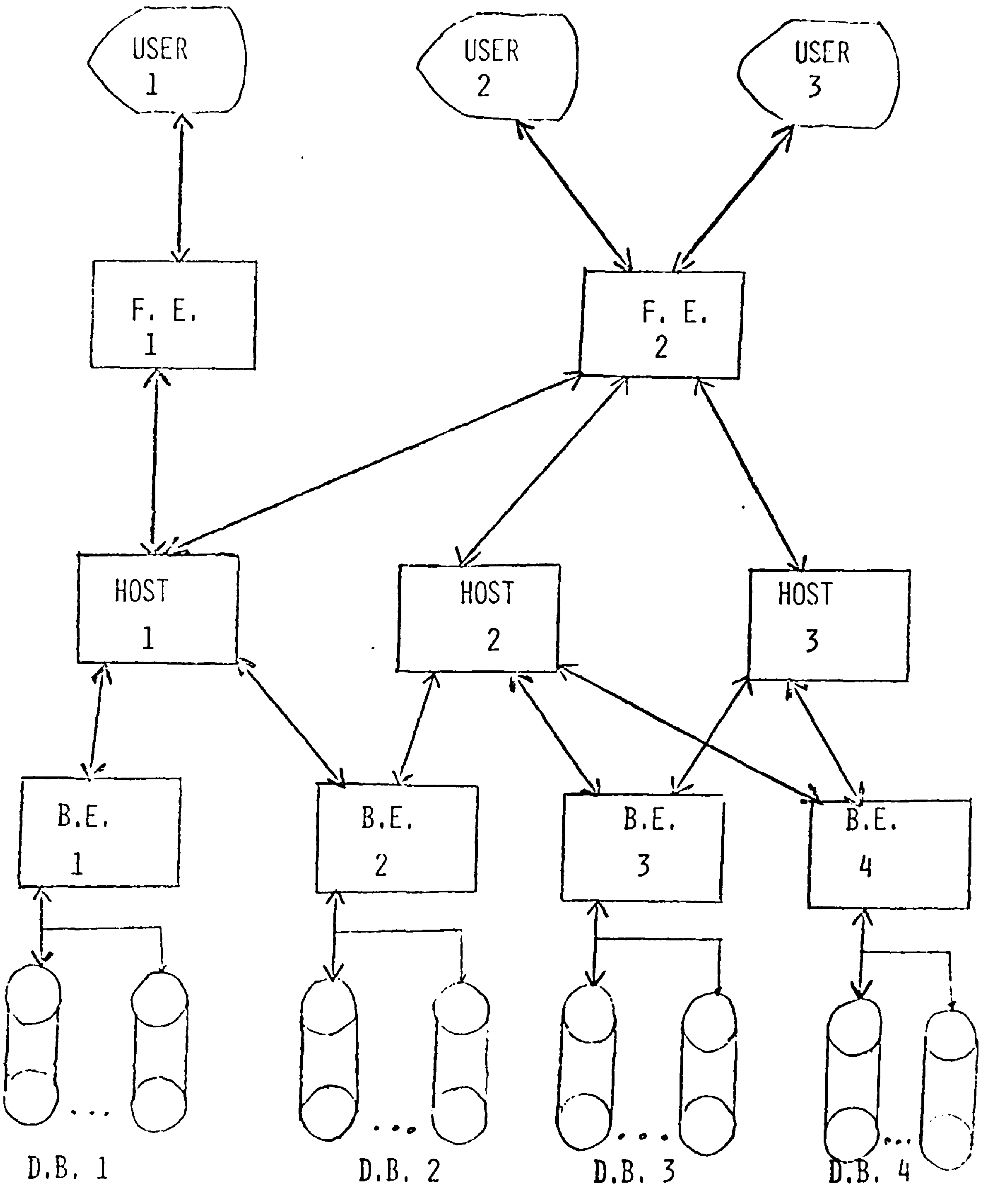

FIGURE 1

DISTRIBUTED DBMS WITH FRONT-END, HOST, AND BACK-END PROCESSORS

8. REFERENCES

1. Fry, J.P. and Sibley, E.H., Evolution of Data Base Management Systems, Computing Surveys, Vol. 8, No. 1, Mar., 1976, pp. 7-42.

2. Aschim, F., Data Base Networks _ _ An Overview, Management Informatics, Vol. 3, No. 1, Feb., 1974, pp. 12-28.

3. Booth, G.M., The Use of Distributed Data Bases in Information Networks, First International Conference on Computer Communication: Impacts and Implications, Oct., 1972, pp. 371-376.

4. Booth, G.M., Distributed Information Systems, National Computer Conference, Vol. 45, June, 1976, pp. 789-794.

5. Canaday, R.H., et al, A Back-End Computer for Data Base Management, Communications of the ACM, Vol. 12, No. 10, Oct., 1974, pp. 575-582.

6. Everest, G.C., The Futures of Data Base Management, ACM SIGMOD Workshop, May, 1974, pp. 445-462.

7. Maryanski, F.J., et al, A Minicomputer Based Distributed Data Base System, NBS-IEEE Trends and Applications Symposium: Micro and Mini Systems, May, 1976, pp. 113-117.

8. Whitney, K.V.M., Fourth Generation Data Management Systems, National Computer Conference, Vol. 42, June, 1973, pp. 239-244.

9. Maryanski, F.J., Fisher, P.S., and Wallentine, V.E., Evaluation of Conversion to a Back-End Data Base Management System, ACM National Conference, Oct., 1976.

10. Denning, P.J., Virtual Memory, Computing Surveys, Vol. 2, No. 3, Sept., 1970, pp. 153-190.

11. Salako, A., A Locality Model for File Structure Organization, Conference on Information Sciences and Systems, Mar., 1976, pp. 194-200.

12. Aho, A.V., Denning, P.J., and Ullman, J.D., Principles of Optimal Page Replacement, Journal of the ACM, Vol. 18, No. 1, Jan., 1971, pp. 80-93.

13. Gelenbe, E., A Unified Approach to the Evaluation of a Class of Replacement Algorithms, IEEE Transactions on Computers, Vol. C-22, No. 6, June, 1973, pp. 611-618.

14. Trivedi, K.S., Prepaging and Applications to Array Algorithms, IEEE Transactions on Computers, Vol. C-25, No. 9, Sept., 1976, pp. 915-921.